

# Bash Basics

Modified from Slides by Erika Hunhoff for CSCI3753  
University of Colorado Boulder

# Table Of contents

1. User Guide
2. Introductory Vocabulary (shells, commands, and scripting)
3. Man Pages
4. Files and File System
5. Permissions and Groups
6. Additional Useful Commands
7. Combining Commands
8. Tips

# USER GUIDE



Work

Work through this slideshow at your leisure



Skip

Skip sections you already know



Pieces

Learn in pieces, don't try to learn everything at once



Experiment

Experiment! Take the time to complete the 'exercise' slides!



Refer

Use as a reference

# What is Shell ?


- A **shell** is an interface one can use to instruct a computer to take some action
- Instructions are typically given in the form of a **command**. Commands can be:
  - A **built-in command** is specific to the shell itself (the code to execute it is part of the shell program)
  - An **external command** runs an external executable. When you call an external command, the kernel or operating system will run the executable and it becomes a process.

# Commands

Commands typically take **arguments**. Arguments are specific to each command.



**Example:** A copy program probably takes a source path (the file to copy) and a destination (the destination of the copy of the file)



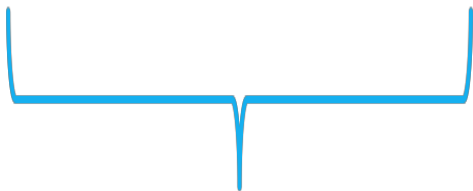
Oftentimes, it's possible to find out information about a specific command by looking at the **man page** (short for 'manual page') for a command (more on this later) or by typing **--help** as the first and only argument

# Scripting

- In addition to commands, shells can also be programmed using a process known as **scripting**.
- Various shells provide a limited set of programming features and can execute commands based on conditions, etc.

# Anatomy of a Shell

```
user@host:/home$ command arg1 arg2 arg3
```



This is the command prompt. When you open a shell, you see this. It may contain some useful information such as your username (`user` in this example), hostname (`host` in example), and current directory (`/home` in example)

# Anatomy of a Shell

```
user@host:/home$ command arg1 arg2 arg3
```



The command that will be run when you select the 'return' or 'enter' key



# Exercise 1: Practice Commands

- `whoami` : your user id
- `pwd` : your present working directory
- `uname -a` : information about the system
- `echo hello` : prints a string to standard out

# Man Pages



Man pages (short for “manual pages”) are a way to learn about various commands and system features



You can read the man page by:

Run: `man <somecommand>`

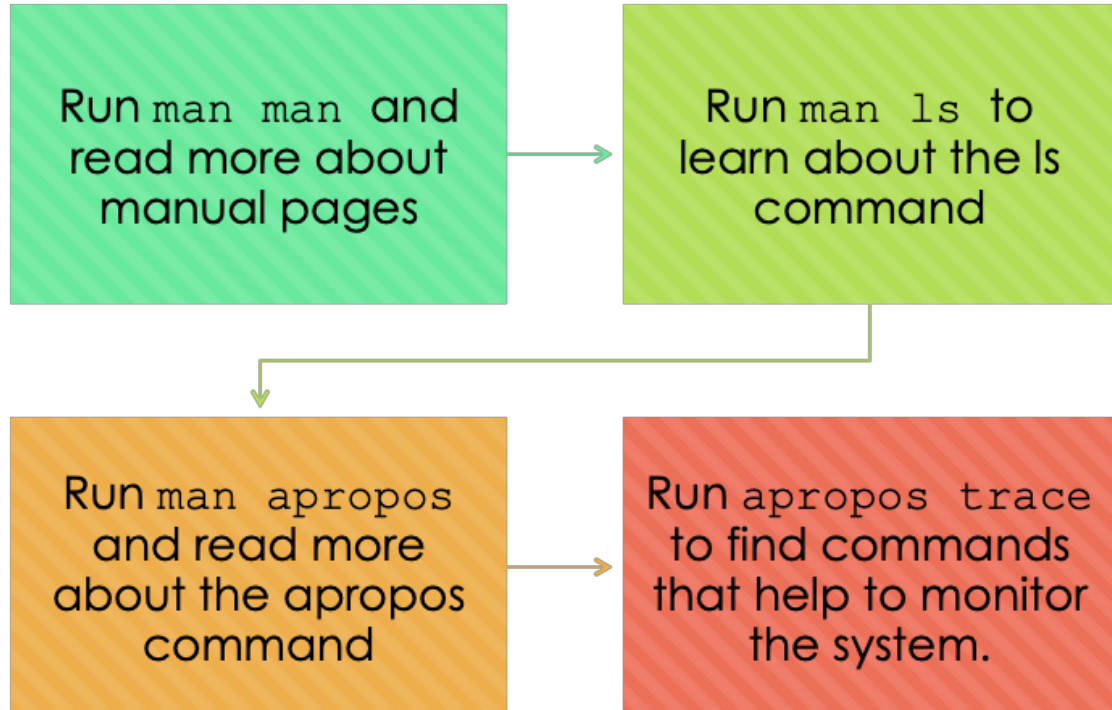
Use arrows to scroll

Press 'q' to quit the man page view and return to your normal terminal.



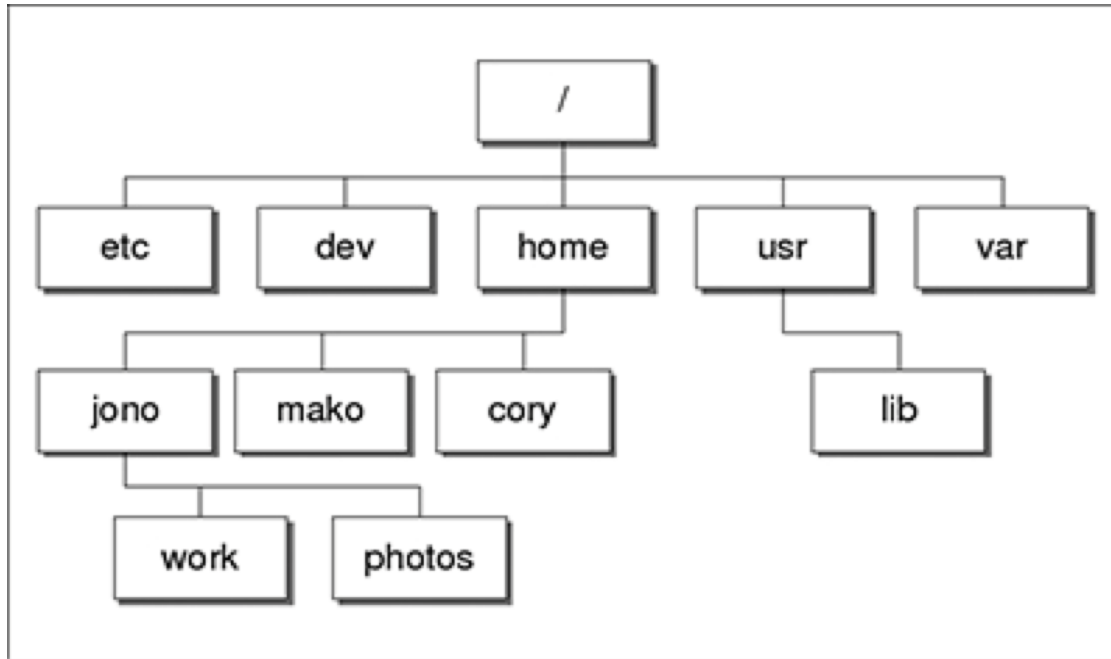
Man pages are organized into sections according to the type of thing being documented. `apropos` is a command to search man pages for a phrase in the name or description

## Exercise 2: Man Pages



# File System and Files

- The Linux file system has a single root, /, and several common directories underneath



## Subdirectories under /

Directory	Content
/bin	Common programs, shared by the system, the system administrator and the users.
/boot	The startup files and the kernel, vmlinuz. In some recent distributions also grub data. Grub is the GRand Unified Boot loader and is an attempt to get rid of the many different boot-loaders we know today.
/dev	Contains references to all the CPU peripheral hardware, which are represented as files with special properties.
/etc	Most important system configuration files are in /etc, this directory contains data similar to those in the Control Panel in Windows
/home	Home directories of the common users.
/initrd	(on some distributions) Information for booting. Do not remove!
/lib	Library files, includes files for all kinds of programs needed by the system and the users.
/lost+found	Every partition has a lost+found in its upper directory. Files that were saved during failures are here.
/misc	For miscellaneous purposes.
/mnt	Standard mount point for external file systems, e.g. a CD-ROM or a digital camera.
/net	Standard mount point for entire remote file systems
/opt	Typically contains extra and third party software.
/proc	A virtual file system containing information about system resources. More information about the meaning of the files in proc is obtained by entering the command <b>man proc</b> in a terminal window. The file proc.txt discusses the virtual file system in detail.
/root	The administrative user's home directory. Mind the difference between /, the root directory and /root, the home directory of the <i>root</i> user.
/sbin	Programs for use by the system and the system administrator.
/tmp	Temporary space for use by the system, cleaned upon reboot, so don't use this for saving any work!
/usr	Programs, libraries, documentation etc. for all user-related programs.
/var	Storage for all variable files and temporary files created by users, such as log files, the mail queue, the print spooler area, space for temporary storage of files downloaded from the Internet, or to keep an image of a CD before burning it.

# Relative vs. Absolute Paths

- In most Linux commands that take a path as an argument, the path may be relative or absolute.
- A **relative path** is the path that is relative to the current directory (pwd)
  - Example: `this/path/is/relative.txt` is equivalent to the value of `<pwd>/this/path/is/relative.txt`
- An **absolute path** is a path that is relative to the root (/) of the file system.
  - Example: `/tmp/this/path/is/absolute.txt`
- The `/` at the front of the path lets you know which kind of path that it is.

# Common File System Commands

rm	Remove
cp	Copy
mv	Move
mkdir	Create a directory
rmdir	Remove a directory
ls	file system listing
touch	updates the modified timestamp of a file (if it exists) or create a new empty file

# ls has a ton of useful options

`ls -a` : show all (including dot files)

`ls -l` : long listing

`ls -al` : long listing of all files

`ls -lh` : long listing in human readable format

`ls -t` : sort based on time

`ls -tr` : sort based on time reverse

`ls -altrh` : string them all together



## Exercise 3: File System Commands

- Test out some commands you've learned:
  - Create an empty file in `/tmp` called `example`
  - List `/tmp` to see your file. Try out the different arguments you can give to `ls` to get a more detailed listing.
  - Copy `example` and name the copy `example2`
  - Delete `example2`
  - Rename `example` to `example3`

# File Commands

- `head` shows top 10 lines
  - `head -N` shows top N lines
- `tail` shows bottom 10 lines
  - `tail -f` follows as the file grows. GREAT for troubleshooting.
- `cat <filepath>` dumps the contents of a file to stdout
- `less <filepath>` is a minimal way to show some portion of a file.
  - `more <filepath>` is also a command.

# File Extensions

- Note: Unlike Windows, Linux is much more casual about file extensions. You don't even need to specify one!
- The `file` command can help you determine the type of file something is, based on its content (specifically header data).

## Exercise 4: File Commands

- Test out some commands you've learned:
  - View the first 15 words in the word list found at `/usr/share/dict/words`
  - View the last 30 words in the word list found at `/usr/share/dict/words`
  - View the entire word list using `less`. Type `/hello` to find all words that contain 'hello' in the list. Quit `less` with `q`
  - Run `file /usr/share/dict/words`. You will realize that the file is not what you thought it was – google 'linux symbolic link' to find out more information. Now run `file <new_path>` until you find the actual location of the file.

# Permissions and Groups

**Permissions** exist to control access to files and directories. `ls -l` will show the permissions of both files and directories:

```
-rw-rw-r-- 1 user user      6 Sep 22 00:33 test.txt
```

# Permissions and Groups

**Permissions** exist to control access to files and directories. `ls -l` will show the permissions of both files and directories:

```
-rw-rw-r-- 1 user user 6 Sep 22 00:33 test.txt
```



**permissions**

Permissions are displayed with three groups of three letters:


r for read, w for write, x for execute.

The first group is for user (owner), the second is for group, and the last is for everyone else.

# Permissions and Groups

**Permissions** exist to control access to files and directories. `ls -l` will show the permissions of both files and directories:

```
-rw-rw-r-- 1 user user 6 Sep 22 00:33 test.txt
```

  
**owner group**

Depending on who you are, and what groups you are in (displayed with the `groups` command), compared to those of the file, you can tell whether you have read, write, and/or execute permission for a given file.

# Permissions and Groups Commands

groups	list the group you (or another user) is a member of
chgrp	changes the group ownership of a file
chmod	changes the file mode bits



# Exercise 5: Setting Permission

- The syntax and format of mode bits can be hard to understand. Google can be a lot of help with `chmod`.
- `touch /tmp/file1`
- Run `ls -l /tmp/file1` to check what the mode bits are for `file1`
- Run `chmod 755 /tmp/file1`
- Run `ls -l /tmp/file1` to check what the mode bits are for `file1`
- Run `chmod 777 /tmp/file1`
- Run `ls -l /tmp/file1` to check what the mode bits are for `file1`
- Run `chmod 640 /tmp/file1man`
- Run `ls -l /tmp/file1` to check what the mode bits are for `file1`

# sudo

- sudo allows users to run command as root without knowing root password.
- The sudoers list lists users who have the permission to run commands with sudo
- su allows you to actually change yourself to a different user. Running su – changes you to root – this can be useful so you don't always have to type sudo but also very dangerous because you can do a lot of damage as the root user

# If you see a permissions error...

- Check the mode bits, owner, and group of the file/directory to make sure you should be able to do what you want to do.
- Once you have identified the problem, common solutions are to:
  - Use `chmod` to change the permissions to allow you to do what you want to the file
  - Use `chgrp` to change the group to the group you have
  - Use `sudo` to override specific user/group constraints.


# Additional Useful Commands

grep	Uses regular expression to search for a text in a file or files
find	Searches the file system for files by a variety of attributes
ps	List the processes running on the machine




# grep

- `grep` is a command to match patterns in strings based on something called regular expressions.
- **Regular Expressions** (or **regex**) is a pattern that is defined by a sequence of characters. Most high-level languages have support for regular expressions, so whether you learn how to use regular expressions using bash, or by using using the Python `re` library, or something else, it's a good idea to learn how to use regular expressions in some capacity.
- Some useful flags for `grep`:
  - `-r` means **recursive**, so if you select a directory it will search for texts in all files under that directory
  - `-n` will print the **line number** the search pattern was found on
  - `-i` indicates that your pattern is **case insensitive**.
  - `-v` negates the pattern (i.e. everything BUT matches)

# grep example

 flags go here

```
grep Bash ~/.bash_profile
```

command    pattern    file to search

## Exercise 6: grep

- Run `man grep` and review the output
- Run `grep --help` and review the output
- Search `/usr/share/dict/words` 'flamingo' and print the line number. Make sure your query is case-insensitive
- Recursively search `/tmp` for the phrase 'user'

# find Command

- Find is a command that searched for files that match certain specified characteristics.
- General format is: `find <location_to_search>  
<flags_or_search_pattern>`
- The next slide has some examples, use the find documentation (`man find` or `find --help`) to try to figure out what each command is doing.



# find Examples

```
find / -name passwd
```

```
find ~ -mtime -1
```

```
find ~ -ctime -1 -name foo
```

```
find /var/log -mtime +7 -exec rm {} \;
```

# Combining Commands & More

- Bash also provides several ways to combine commands
- You can **pipe** output from one command to another using the `|` operator.
- You can **chain** commands (run more than one command on one line) with `;` or `&&`. `;` runs each command independently, without checking return values. On the other hand, `&&` will only proceed to the next command if the first succeeded.

# | operator

An example of using the | operator is:

```
cat /usr/share/dict/words | grep hi | wc -l
```

The first section (`cat`) will print the contents of `/usr/share/dict/words` to `stdout`. The first | will send that output to the `grep` command, which will search the text for the string 'hi'. The last pipe will send all the words that contain 'hi' to `wc -l`, which will count the number of lines in the output.

The end result is a command of how many words contain the substring 'hi' in the dictionary.

# &&, ||, and ;

An example of chaining commands is:

```
cd /bin; ls -lah
```

This allows you to change directories and do a listing with a single command. Of note, the second command will *a/ways* be run, even if the first one fails. You could also run:

```
cd /bin && ls -lah
```

This command will do the same thing. However, the second command will *only* run if the first is successful. As a note, || does the opposite thing as &&: with || the second command will only run if the first command is unsuccessful.

# Exercise 8: Chaining & Combining

- Count the number of files in `/etc`, use `| wc -l`
- Get a list of all files in `/bin` with `bash` in the name
- Get a count of the above

# Running in the Background

You may have noticed that long-running commands rob your ability to use a terminal. For instance, if you type `gedit` to open `gedit` in a terminal, you won't have control of the terminal again until `gedit` is closed.

To fix this there are two options:

- Start the program as a background process with `&`. For example, run `gedit &`
- Send the process to the background. First, suspend execution with `ctrl+Z`. Then, type `bg`

# Tips

There are a lot of useful tips and tricks that can increase your efficiency (and productivity) when working with bash. Here are a few of them:

- Learn how to open the terminal with several keystrokes (instead of opening it up through a GUI menu). On most systems it's typically something like `ctrl + T` or something similar.
- Bash can be configured using the `.bashrc` file in your home (`~`) directory. Here, you can set environment variables and also alias commands that might be useful to you. Google can help you with this!
- You can tab complete files names when using many commands such as `cd`, `ls`, and more
- You can view command history (commands you've previously run) using the up arrow key
- There are commands to move your cursor to the end or beginning of a command line. This can be really helpful if you type a long command and then realize you messed up the first character... definitely worth learning!



## Thank You!

Hopefully you found this useful! There are a great many commands and it's simply not feasible to cover them all. The commands here will give you a great start, however. Bash is almost always overwhelming when you are a beginner – don't try to learn everything at once, but try to get familiar with 2-3 commands per week. If you learn a little at a time consistently, you will become an accomplished user before too long!